Control de tráfico y aspectos de seguridad Usando herramientas OpenSource

Leonardo de- Matteis[*], Javier Echaiz[**], Jorge R. Ardenghi

Laboratorio de Investigación de Sistemas Distribuidos (LISiDi)
Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur, Bahía Blanca (8000), Argentina
{ldm,je,jra}@cs.uns.edu.ar

Resumen

Hoy en día, gran cantidad de organizaciones carecen de políticas claras de seguridad en sus redes y sistemas internos. Sus intranets están diseñadas y administradas sin considerar aspectos de seguridad como el control de tráfico. Los administradores suelen considerar que este último consiste en implementar prohibiciones ante problemas surgidos con algún usuario de la red y se decide prohibir, por ejemplo, la utilización de programas peer-to-peer por diversas razones: uso de ancho de banda excesivo, piratería, intercambio de información privada interna, etc. El control de tráfico y la calidad del servicio, son conceptos independientes. Los administradores deben tener en claro que las políticas de seguridad abarcan tanto control de tráfico como calidad de servicio. Esta última será la que percibirá el usuario al acceder a la red desde su estación de trabajo, cuando visualice una página web o reciba su correo en forma rápida y ágil, o bien cuando reciba transmisiones de video lentas, cortes y demoras en servicios no primordiales. Aquellas políticas implementadas para controlar el tráfico sólo serán percibidas por los usuarios cuando no puedan acceder a ciertos servicios de la red, o cuando traten de utilizar programas no permitidos, y vean que las conexiones no pueden establecerse o son muy lentas. El objetivo de este trabajo es señalar aspectos a considerar al implementar políticas de seguridad que incluyan control del tráfico entrante y saliente de la red interna de una organización.

Palabras claves: seguridad, control de tráfico, calidad de servicio, QoS, firewall.

[*] Becario Agencia Nacional de Promoción Científica y Técnica (ANPCyT), PICT 15043, Argentina.
[**] Becario del Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET), Argentina.

1. INTRODUCCIÓN

Planteamos como escenario de trabajo una organización que tiene acceso directo a Internet. Se buscará alcanzar una mayor seguridad y mejor calidad de servicio dentro la organización, implementando controles utilizando un equipo con sistema operativo GNU/Linux. Más allá de implementar simplemente un *router* con un *firewall* asociado, se buscará diferenciar el tráfico, proteger el acceso desde y hacia la intranet y aplicar ciertos parámetros de calidad de servicio, para que la percepción del usuario sea la correcta y para que pueda realizar sus tareas en forma apropiada.

En un escenario así definido, se tendrán en cuenta los siguientes requerimientos para implementar las políticas de protección en el equipo *router/firewall* [1]:

- El equipo que cumple la función de *router* estará protegido por un *firewall* propio y, a su vez, protegerá los equipos que se encuentran detrás del mismo.
- Se negará el acceso a servicios que corran en el *router*, excepto a los estrictamente necesarios.
- Sólo se permitirá el reenvío de paquetes a través del *firewall* desde y hacia la intranet para los puertos permitidos según las políticas de seguridad a implementar.

Por otro lado, para establecer las políticas de control de tráfico, se considerarán los siguientes requerimientos:

- brindar alta disponibilidad para servicios como: POP3, SMTP, SSH, HTTP, HTTPS;
- proteger servicios internos de la intranet, no permitiendo su acceso desde el exterior, por ejemplo: servidores MySQL, RPC, etc.;
- evitar el egreso de información fuera de la red de la empresa (ejemplo: keyloggers, troyanos, programas de análisis de preferencias, etc.), impidiendo forwarding de paquetes de puertos no utilizados;
- implementar un proxy transparente (tráfico HTTP);
- brindar niveles de calidad de servicio a cada uno de los equipos de la intranet, según cada caso, para así poder controlar el upload y download máximo permitido en cada uno.

Se detallarán los lineamientos básicos para implementar en próximos trabajos la detección de tráfico particular, teniendo en cuenta los siguientes objetivos:

- detectar el tráfico asociado a programas de intercambio de archivos (P2P);
- negar por defecto el tráfico P2P, y prever la posibilidad de su utilización sólo a algunos equipos de la intranet;
- disminuir en horarios específicos el consumo de ancho de banda por parte de programas P2P o bien restringirlo completamente.

Para la implementación de las funciones de *router*, *firewall*, control de tráfico y calidad de servicio, se utilizarán los siguientes componentes: iproute2 (ip, tc), iptables, tcng, php y bash.

[1] En lo que sigue, las denominaciones *router/firewall* harán referencia al mismo equipo.

2. CONCEPTOS PARA LA IMPLEMENTACIÓN

Bajo el sistema operativo GNU/Linux, cada tarjeta de red (NIC: *Network Interface Card*) posee un controlador asociado, denominado *network driver*, el cual controla la comunicación con el hardware. Dicho controlador posee un mecanismo de intercambio de paquetes que se ubica entre el código para gestión de red del sistema operativo y la capa física propiamente dicha de la misma. Podemos apreciar en el siguiente esquema en qué punto se ubica al controlador del dispositivo:

Ahora veremos conceptos básicos para entender cómo se implementa el control de tráfico sobre GNU/Linux. Dicho control se basa en cuatro componentes que explicaremos brevemente a continuación, en una breve introducción a lo que se denomina TC (*Traffic Control over Linux*) [2], [9], [6].

2.1. Disciplinas para el manejo de colas

Las disciplinas para el manejo de colas son algoritmos para controlar colas asociadas a los dispositivos de red, permiten especificar la forma de manipular los paquetes en las mismas y a su vez crear nuevas. Estas se encuentran en áreas de memoria del sistema operativo y tienen un tamaño determinado de acuerdo a los paquetes que se almacenarán en ellas y otros aspectos de performance. El algoritmo asociado a una determinada cola y la cola propiamente dicha (como estructura de datos) los asociaremos como una unidad, que denominaremos de ahora en más con la abreviatura *qdisc* (*queuing disciplines*). Cada dispositivo tiene asociada una cola de ingreso y otra de egreso. En la documentación disponible sobre el sistema operativo GNU/Linux, la primera se denomina *ingress* y la segunda *egress*.

Existen 11 tipos de disciplinas soportadas bajo el sistema operativo GNU/Linux, cualquiera de ellas es aplicable a un *qdisc*, de acuerdo al problema a resolver [13] y [5]:

- Class Based Queue (CBQ)
- Token Bucket Flow (TBF)
- Clark-Shenker-Zhang (CSZ)
- First In First Out (FIFO)
- Priority Traffic Equalizer (TEQL)
- Stochastic Fair Queuing (SFQ)
- Asynchronous Transfer Mode (ATM)
- Random Early Detection (RED)
- Generalized RED (GRED)
- Diff-Serv Marker (DS-MARK)
- HTB (Hierarchical Token Bucket)

2.2. Terminología

Root qdisc: Un *qdisc* raíz es la que por defecto se encuentra adjunta a un dispositivo de red.

Qdisc sin clases: Un *qdisc* sin subdivisiones internas configurables. Es necesario destacar en este punto y en lo que sigue que, según el contexto de uso de *qdisc*, el mismo puede hacer referencia a la estructura de datos "cola", a la disciplina de encolamiento asociada a la estructura de datos o, inclusive, al conjunto de ambos.

Qdisc con clases: Un *qdisc* que puede contener múltiples clases y, a su vez, estas pueden contener otras *qdisc*. No todas las disciplinas de encolamiento permiten el uso de clases.

Clases: Las clases se pueden definir como propiedades de los *qdisc*. Las clases no almacenan paquetes, es decir, éstas a su vez utilizan otros *qdisc* para ello. La disciplina de encolamiento para un

determinado *qdisc* se puede elegir de un conjunto de diferentes disciplinas y, por lo tanto, estos *qdisc* pueden a su vez tener diferentes clases. Entonces se establece un árbol de jerarquías, los *qdisc* tienen clases, entonces éstas a su vez pueden contener otros *qdiscs*. Y estos últimos pueden o no tener a su vez otras clases, que utilicen otras disciplinas de encolamiento.

Filtros: Por otro lado, es una necesidad poder clasificar los paquetes, para ello necesitamos contar con diferentes tipos de filtros. En el sistema operativo, cada dispositivo de red tiene por defecto una disciplina de encolamiento, por lo tanto, tiene un *qdisc* asociado predeterminado. De esta manera se controla cómo los paquetes son encolados en ese dispositivo. Si es necesario poder distinguir diferentes clases de paquetes que pasan por el dispositivo entonces necesitamos utilizar estos filtros, que son el tercer componente para la implementación del control de tráfico.

Políticas: Se utilizan para retrasar o descartar paquetes, permitiendo que el tráfico se mantenga por debajo de un ancho de banda configurado. Sólo se pueden descartar paquetes, no retrasarlos, ya que no se cuenta con una cola de ingreso que permita el retraso de los paquetes que se van recepcionando (se debe tener en cuenta que no se puede disminuir la velocidad con la que otro equipo envía los paquetes por la red hacia el equipo destino).

2.3. Disciplinas de encolamiento utilizadas

2.3.1. Hierarchical Token Bucket (HTB)

Esta disciplina de encolamiento permite controlar el uso de ancho de banda de un determinado enlace. Usando esta técnica podemos simular, sobre un único enlace físico, la existencia de enlaces más pequeños, y enviar diferentes tipos de tráfico por ellos. Al utilizar esta disciplina debemos determinar primero qué enlaces simulados necesitamos, las restricciones sobre los mismos y los tipos de paquetes que vamos a distribuir sobre cada uno. Este algoritmo nos permite contar con un método más intuitivo y entendible, sin demasiados parámetros que entorpezcan nuestras apreciaciones al momento de implementar el control de tráfico deseado. Dentro de esta disciplina tenemos una serie de parámetros claves, que permiten controlar la cantidad de ancho de banda que se suministra a cada clase, el cual debe ser al menos de la cantidad que la misma requiere y de la cantidad que le fue asignada.

Asociados a cada *qdisc*, podemos definir parámetros para el control de ancho de banda sobre dicho tipo de encolamiento. A continuación detallamos algunos de esos parámetros, los cuales se utilizarán en nuestra implementación:

Rate: permite definir el ancho de banda asignado y garantizado para un determinado *qdisc*. Es decir, si definimos para un *qdisc* un *rate* de 128kbit/s, entonces se está determinando el mínimo ancho de banda con el cual contará el mismo.

Rate ceiling: es la máxima cantidad de ancho de banda que un *qdisc* puede tener. Si, por ejemplo, el mismo es de 256kbit/s, entonces un cliente asociado a dicho *qdisc* posee un máximo de 256kbit/s, para su tráfico entrante (tráfico saliente por el dispositivo que tiene asociado dicho *qdisc*).

Burst: es el tamaño del contenedor (*bucket*) utilizado para el control de ancho de banda por unidad de tiempo, es decir, el parámetro *rate*. Bajo la disciplina HTB, se sacará de la cola la cantidad de bytes asignados según el parámetro *burst*, antes de esperar el arribo de nuevos tokens.

CBurst: es el tamaño del contenedor (*bucket*) utilizado para el control del máximo ancho de banda por unidad de tiempo, es decir el parámetro *rate ceiling*. Bajo la disciplina HTB, se sacarán de la cola la cantidad de bytes asignados según el parámetro *cburst*, antes de esperar el arribo de nuevos tokens.

Prio: permite asignar prioridades a las clases. Un número menor implica una prioridad mayor. Este parámetro es utilizado, por lo general, cuando las clases intentan obtener un ancho de banda mayor que el garantizado en el *qdisc* asignado a la clase. Así se puede definir un mecanismo de elección para determinar qué clase gana el ancho de banda disponible en un instante de tiempo.

Se debe tener en cuenta que esta disciplina permite compartir ancho de banda entre diferentes clases, pero la distribución (*bandwith shaping*) propiamente dicha del mismo, ocurre solamente en las hojas (*leaf classes*) del árbol asociado a una determinada interfase. En el mejor de los casos se debería verificar la siguiente regla: "la suma de los *rates* de las hojas no debería exceder el valor del *rate ceiling* para la clase padre que las contiene, es decir debería ser a lo sumo del mismo valor". Así se puede distribuir el ancho de banda no usado en forma correcta entre los *qdisc* presentes en las hojas. Con este propósito existe un parámetro más, que se configura en forma automática, el *quantum*, que se utiliza cuando el *rate* real utilizado en una determinada clase está por encima del *rate* y debajo del *ceil* especificados para la misma. El *quantum* por defecto se configura con el valor de MTU (*Maximun Transfer Unit*) de la interfase, o un valor mayor. Nunca debe configurarse en un valor menor, ya que el algoritmo no sería capaz de calcular precisamente el ancho de banda real consumido en cada momento.

2.3.2. Stochastic Fair Queuing (SFQ)

Las sesiones TCP o los flujos UDP pueden ser vistos como conversaciones, cada una con su cola respectiva. Para permitir que todas las conversaciones reciban un tratamiento igualitario y ninguna predomine sobre las otras, se puede utilizar un algoritmo de "colas justas" como el SFQ. El mismo administra el tráfico con una política de *round robin*, dando a cada conversación (sesión/flujo) turnos para enviar sus datos. De esta manera se tratan en forma uniforme, sin necesidad de configuraciones adicionales. Cada paquete recibido en la cola de entrada es asignado a un contenedor, para esto se utiliza una fórmula *hash* que recibe como parámetros los siguientes valores:

```
(<dirección de origen>, <dirección de destino>, <puerto de origen>)
```

Con estos datos se deriva el paquete al contenedor calculado con la función de *hash*. Múltiples sesiones o flujos pueden ser asignados al mismo contenedor, por lo que el algoritmo de *hash* puede ser perturbado a intervalos configurables. Así se evita que una conversación predomine sobre otra. Luego los paquetes se van retirando de cada contenedor y se van encolando mediante una cola de salida con una política de *round robin*.

Hay dos parámetros que pueden modificar el comportamiento de esta disciplina de encolamiento:

Perturb: intervalo en segundos para "perturbar" el algoritmo de encolamiento que utiliza una función *hash*. El valor por defecto es 0, lo cual significa que no habrá ningún cambio en la forma de asignar los paquetes en los contenedores. En la documentación de la implementación de la SFQ el valor recomendado es 10.

Quantum: cantidad de paquetes de un flujo o sesión que se permite sacar de la cola antes de que le toque el turno a la siguiente cola, es decir, durante una iteración del algoritmo de round robin. El valor es la cantidad de bytes definida por el parámetro MTU de la interfaz asociada, es el mínimo valor permitido, no debe ser inferior.

3. DESARROLLO DE LA PROPUESTA

Para la implementación del control de tráfico propuesto, se optó por utilizar soluciones provistas por herramientas de software libre, para lo cual se estudiaron detalles de operación de las mismas. En la actualidad se pueden implementar soluciones de control del tráfico utilizando la herramienta tc (*Traffic Control*) escrita por Alexey N. Kuznetsov, la cual permite manipular y mostrar las configuraciones que el sistema operativo posee en cuanto al manejo de los dispositivos de red, así como el flujo de paquetes a través de los mismos. En un principio, al usuario que por primera vez necesita

establecer soluciones de control de tráfico vía TC, puede resultarle difícil el estudio e implementación de la misma, pero a pesar de esto es la más difundida y utilizada hoy en día. Luego de un tiempo de uso, se comienza a entender en forma clara y precisa cada uno de los componentes involucrados en una red, y cómo se lleva a cabo el control de tráfico bajo el entorno GNU/Linux.

Una desventaja es el escalamiento de políticas de control especificadas con TC. Si la solución se debe aplicar a redes con gran cantidad de equipos y consideraciones, se hace muy difícil mantener los *scripts* de configuración, los cuales crecen en tamaño y especifican una estructura de árbol jerárquico cuya envergadura se complica en forma excesiva (por ejemplo: la numeración de los nodos del mismo). Por otra parte, el uso de filtros para poder establecer prioridades sobre cada tipo de tráfico agrega aún mayor complejidad, tanto en el *script* de configuración como en las tareas necesarias para contar con un control estadístico sobre cada uno. Se dificulta entonces un mantenimiento fácil de estos *scripts*, sin cometer errores en las sucesivas modificaciones sobre los mismos durante el transcurso del tiempo.

Como contrapartida a tales desventajas, en la actualidad el acceso a este tipo de soluciones basadas en software libre, sin costo alguno más allá del tiempo del tiempo de aprendizaje y configuración del sistema, posibilita el uso de estos recursos a empresas PyMEs, y aun a otras de mayor envergadura. Este tipo de soluciones conlleva una ventaja económica muy grande, ya que adquirir hardware específico, diseñado para satisfacer estas mismas necesidades involucra costos muy altos que, por lo general, son imposibles de afrontar en el mercado comercial argentino actual.

Teniendo en cuenta todas estas consideraciones, la propuesta de trabajo planteada incluye utilizar la herramienta denominada *tcng* (TC *Next Generation*), escrita por Werner Almesberger, y una combinación de *scripts* realizados con el lenguaje php, para lograr la automatización de la generación de las configuraciones necesarias para el control de tráfico propuesto.

Al utilizar *tcng*, podemos escribir los mismos *scripts* hechos con el comando tc pero, al utilizar el meta lenguaje que nos brinda *tcng*, podemos describir la misma semántica que alcanzábamos con los comandos tc en forma más sencilla. Para mayor información sobre la forma de trabajo de *tcng* se recomienda realizar una lectura inicial de [8] y recurrir a la documentación oficial de su creador en [1], en la cual se encuentran muchos ejemplos que ayudan al lector en su aprendizaje y entendimiento de las equivalencias con los comandos de configuración de la herramienta tc. Por otra parte, *tcng* prevé la incorporación de módulos a medida, para complementar el código del kernel y así ampliar su funcionalidad.

Para la implementación de nuestra propuesta se utilizan los siguientes componentes, con los cuales se alcanzan los objetivos propuestos:

1. Escritura de las reglas de para un *firewall* básico según las necesidades de la organización, utilizando un *script* clásico del intérprete de comandos *bash* sobre GNU/Linux.
2. Generar automáticamente a partir de un archivo de configuración:

    a) las reglas de *tcng* necesarias para el control de tráfico entrante (download) hacia los equipos de la intranet;
    b) las reglas de *tcng* necesarias para el control de tráfico saliente (upload) desde los equipos de la intranet;
    c) las reglas de iptables necesarias para el marcado de paquetes necesario para el control del tráfico saliente;
    d) las reglas de iptables para establecer contadores para monitoreo;
    e) los archivos de consulta para poder efectuar un monitoreo de los *qdisc* asociados a cada regla de control por equipo;

3. Programación de un *script* php que genere una salida jerárquica con detalles de los *qdisc* de cada interfase, su tráfico y parámetros asociados, para poder monitorear en tiempo real el comportamiento de las reglas especificadas.
4. Programación de un programa utilizando lenguaje C que permita obtener los datos del *script* en php y mostrarlos por consola, para el caso en que el operador del sistema desee consultarlos de esa forma.

4. IMPLEMENTACIÓN

A continuación se detallarán aspectos sobre la implementación, la respectiva funcionalidad y los objetivos que se cubren con cada uno de los componentes del software, de los que se especifican las versiones utilizadas al probar la implementación desarrollada:

- iproute2 (versión 2.6.9) http://developer.osdl.org/dev/iproute2/download/
- iptables (versión 1.2) http://www.netfilter.org
- tcng (versión 10b) http://tcng.sourceforge.net

Los paquetes correspondientes de iproute2 e iptables suelen encontrarse en todas las distribuciones de Linux en la actualidad. Para la utilización de *tcng*, se recurrió al sitio oficial, se bajaron los fuentes y se compiló sobre la distribución Linux CentOS Server v4.2, con kernel versión 2.6.9.22, utilizado para las pruebas durante la desarrollo y testeo.

4.1. Firewall

Las reglas del *firewall* contemplan los siguientes aspectos de seguridad sobre el *router* y reglas de redirección/paso desde y hacia a la red interna[2]:

- Política por defecto DROP para la cadena INPUT (paquetes dirigidos al equipo).
- Política por defecto ACCEPT para la cadena OUTPUT (paquetes dirigidos al equipo).
- Política por defecto DROP para la cadena FORWARD (derivación de paquetes no dirigidos al equipo).
- Si la interfase pública se encuentra activa, se especifican las siguientes reglas en la cadena INPUT:
    - Permitir el acceso desde ciertas redes confiables al servidor de administración remota SSH, puerto 22 TCP.
    - Permitir el acceso al puerto de recepción de correo electrónico para envíos vía SMTP, puerto 25 TCP.
    - Permitir el acceso a la consulta el servidor DNS de la organización, puerto 53 TCP/UDP.
    - Permitir el acceso a las páginas del servidor web local, puerto 80 TCP.
    - Permitir el acceso al puerto de consulta de correo electrónico protocolo POP3, puerto 110 TCP.
    - Permitir el ingreso de paquetes de consulta de hora, protocolo NTP (Network Time Protocol), puerto 123/TCP.
    - Permitir el ingreso de paquetes dirigidos a los puertos de usuario [1024, 65535] TCP/UDP.
    - Permitir el ingreso de paquetes correspondientes al protocolo ICMP (Internet Control Message Protocol), que tienen sentido en la interface pública del *router/firewall* que se implementa. Tipos aceptados: (8)echo request, (0)echo replay, (3)destination unreachable, (11)time exceeded, (30)traceroute.
- Sobre la interfase privada se especifican las siguientes reglas en la cadena FORWARD:
    - Permitir el acceso a servidores FTP (vía protocolo activo), puertos 20/21 TCP.
    - Permitir el acceso a servidores remotos vía SSH puerto 22 TCP.
    - Permitir el acceso al puerto de recepción de correo electrónico protocolo SMTP (internos y externos), puerto 25 TCP.

[2]Para una mejor comprensión se recomienda consultar el manual de iptables y grafo de estados correspondiente por los cuales pasan los paquetes [4]

- Permitir el acceso al puerto de recepción de correo electrónico para envíos vía SMTP-AUTH (internos y externos), puerto 587 TCP.
- Permitir el acceso a la consulta sobre DNS, puerto 53 TCP.
- Permitir el acceso a sitios web internos y externos, protocolos HTTP y HTTPS, puerto 80 y 443 TCP respectivamente.
- Permitir el acceso al puerto de consulta de correo electrónico protocolo POP3 e IMAP (internos y externos), puerto 110 y 143 TCP.
- Permitir el acceso a puertos programa IRC, puertos [6667,6669] TCP.
- Permitir el acceso al puerto SSL SMTP de Google, puerto 465 TCP.
- Permitir el acceso al puerto SSL POP3 de Google, puerto 995 TCP.
- Permitir el ingreso de paquetes dirigidos a los puertos de usuario de las máquinas de la intranet, puertos [1024, 65535] TCP/UDP.
- Permitir el ingreso y egreso de paquetes correspondientes al protocolo ICMP. Tipos aceptados: (8) echo request, (0) echo replay.

- Sobre la interfase privada se define una regla de preruteo para implementación de un proxy transparente (utilizando Squid 2.5), destino puerto 80 redirección a puerto 1880.
- Sobre la interfase pública se especifica una regla de SNAT, para mapear todos los equipos de la intranet y sus direcciones al IP público sobre dicha interfase.

4.2. Control de tráfico y priorización

A continuación se describen los componentes que permiten implementar el control de tráfico deseado para la organización. Asumimos que los archivos involucrados se encuentran en la siguiente ubicación: `/etc/tcng/`. Para la implementación se tiene como base un conjunto de reglas definidas por IP. Dichas reglas se encuentran ubicadas en el archivo `tcng_limits.txt` y tienen la siguiente sintaxis:

```
<nombre_equipo>; <IP[,IP]>;<rate_maximo_por_ip>; <rate_maximo_total>
```

Donde `<nombre_equipo>` es un alias asignado para identificar el cliente o los equipos del mismo, asociado(s) al IP(s), `<IP>` es el número de IP o los números (separados por comas) de los equipos clientes, `<rate_maximo_por_ip>` es el *rate* máximo asignado a cada IP, `<rate_maximo_total>` es el *rate* máximo total global para todos los IPs del cliente, el cual funciona como límite *ceil*.

La medida asociada al *rate* especificado es kilobits, ejemplos:

```
cliente1;10.0.10.13;128;128
cliente2;10.0.10.27,10.0.10.28;64;128
cliente3;10.0.10.30;128;256
```

En el caso del cliente 1, posee un *rate* de 128kbps para su IP y un *ceil* máximo de 128kbps, es decir, se le suministra como máximo un ancho de banda de 128kbps. Para el cliente 2 cada IP tiene al menos un *rate* de 64kbps, con un *ceil* de 128, es decir que si alguno de los equipos del cliente 2 no está utilizando dicho ancho de banda, el mismo es aprovechado por los restantes. Para el cliente 3 el *rate* es de 128kbps, siendo el *ceil* 256kbps, por lo que el cliente tiene un ancho de banda como mínimo de 128kbps, si se encuentra disponible globalmente.

El proceso que genera las reglas *tcng* y las de marcado de paquetes vía iptables necesarias para el control de tráfico consiste en:

1. Obtener las reglas desde el archivo de límites: `tcng_limits.txt`.
2. Generar las reglas *tcng* para la interfase interna (ver en la figura 1 la interfase interna eth1 del *firewall/-router*), control de download desde los clientes en la intranet.
3. Generar las reglas *tcng* para la interfase externa (ver en la figura 1 la interfase externa ppp0 del *firewall/-router*), control de upload desde los clientes en la intranet.
4. Generar el archivo: `tcng_rules.txt`, con la configuración *tcng* obtenida a partir de los conjuntos de reglas obtenidos en (2) y (3).

5. Compilar las reglas a partir del archivo obtenido en (4), y se obtiene la configuración de comandos tc para configurar el kernel posteriormente. En este paso se obtiene el archivo `tc_locations.txt` con las descripciones necesarias para implementar el marcado de paquetes utilizado para el control del tráfico saliente. El archivo con la configuración vía comandos tc que se obtiene es `tc_config.txt`.
6. A partir del archivo de descripciones, en este paso se generan las reglas para el marcado de paquetes, destinado al control del tráfico saliente vía iptables. El archivo obtenido es un *script* en lenguaje *bash*: `upload_control_rules.sh`.
7. Generar las reglas para poder graficar con la herramienta sasacct el ancho de banda utilizado por cada uno de los clientes y sus números de IP asignados.
8. Por último, se generan dos archivos de descriptores (eth1.desc y ppp0.desc), cada uno de los cuales contiene un arreglo (en lenguaje php) que indica el *qdisc* asignado a cada cliente/IP(s). Estos archivos son utilizados por las herramientas de monitoreo en tiempo real de los *qdisc* de cada interfase involucrada en el control de tráfico (en este caso ppp0 y eth1, ver en la figura 1 el grafo de la red).

Todo este proceso se lleva a cabo automáticamente ejecutando el *script* `generar.php` de la siguiente forma: `php -q generar.php`, luego de lo cual se obtienen todos los archivos citados en el mismo directorio de trabajo `/etc/tcng/`.

Ejemplo de salida obtenida para configuración de reglas *tcng* [1] de un cliente en particular, control de tráfico entrante (download):

```
dev eth1 {
  egress {
    class (<$clienteA_máquina0>) if ip_dst == 10.0.10.27;
    class (<$clienteA_máquina1>) if ip_dst == 10.0.10.28;
    htb() {
      class ( rate 1024kbps, ceil 1024kbps ) {

        class ( rate 128kbps, ceil 128kbps ) {
        $clienteA_máquina0 = class ( rate 64kbps, ceil 128kbps, tag "ifi.clienteA_máquina0") {
          class ( prio 1) on rsvp element
            (ipproto "tcp",dst 10.0.10.27, src 10.0.0.2) {sfq (perturb 10s);};
          class ( prio 2) on rsvp element
            (ipproto "tcp",dst 10.0.10.27) {sfq (perturb 10s);};
        };

        $clienteA_máquina1 = class ( rate 64kbps, ceil 128kbps,tag "ifi.clienteA_máquina1") {
          class ( prio 1) on rsvp element
            (ipproto "tcp",dst 10.0.10.28, src 10.0.0.2) {sfq (perturb 10s);};
          class ( prio 2) on rsvp element
            (ipproto "tcp",dst 10.0.10.28) { sfq (perturb 10s); };
        };
      }
    } // fin clase (principal) } // fin htb } // fin egress
} // fin dev
```

El control de tráfico implementado tiene en cuenta que todo el tráfico con origen en el mismo *router* tendrá mayor prioridad que el tráfico de download proveniente desde la red externa (cadena FORWARD de iptables). De esta manera, los clientes pueden estar usando todo su ancho de banda, pero perciben que los servicios de POP3, SMTP, DNS y otros similares, implementados sobre el *router* mismo, no se ven afectados, por ejemplo, si está bajando mucha información vía FTP o con programas P2P, que hacen que el ancho de banda asignado al cliente esté totalmente ocupado.

Como se puede observar, hay dos clases con prioridades diferentes, que implementan los *qdisc* respectivos. En la primera clase con prioridad: prio 1, podemos ver que se utilizan los filtros disponibles en *tcng*, en este caso rsvp, con el cual se filtran todos los paquetes que tienen IP 10.0.10.14 como destino, y el IP 10.0.0.2 como origen, es decir que provienen de un IP de la interfase interna (eth1) del mismo *router*. La segunda clase, con prioridad: prio 2, sirve para considerar todo el tráfico restante destinado al IP 10.0.10.14 del cliente.

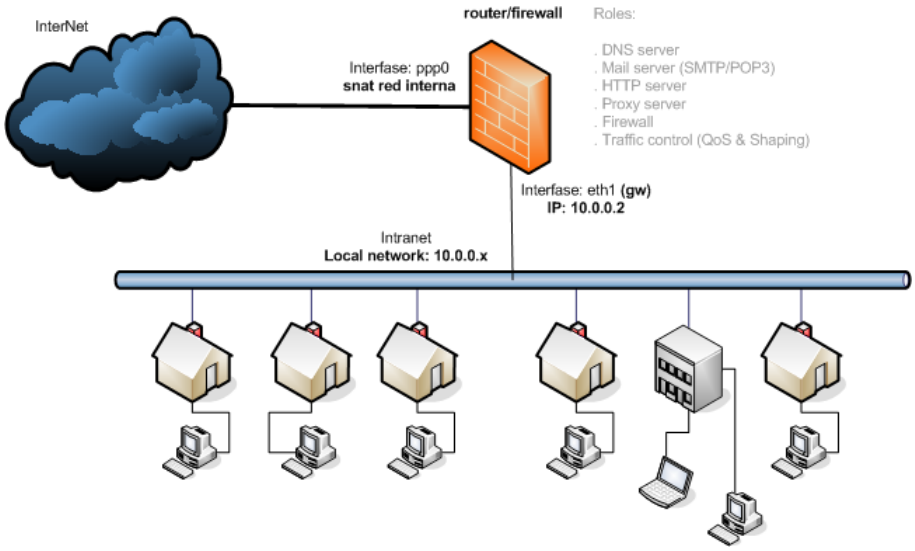

Figura 1: Topología de la red utilizada para la implementación.

System: up 22 days, 18:22, 0 users, load average: 0.08, 0.04, 0.01 | **Now:** Mon July 17 2006 @ [redacted] | **Device:** eth1

class	rate	actual rate	ceil	transfer	type	qdisc	prio	client	% of total
2:1	**1024Kbit**	681.6 Kbit/s	1024Kbit	10067242 KB	root	htb			**66.56**
2:12	64Kbit		128Kbit	10585 KB	leaf	htb	0	[redacted]	0.00
2:13	64Kbit	2.7 Kbit/s	128Kbit	573180 KB	leaf	htb	0	[redacted]	0.26
2:14	64Kbit		64Kbit	669170 KB	leaf	htb	0	[redacted]	0.00
2:15	64Kbit		128Kbit	725 KB	leaf	htb	0	[redacted]	0.00
2:1d	64Kbit	64.6 Kbit/s	64Kbit	102571 KB	leaf	htb	0	[redacted]	0.00
2:17	64Kbit	2.0 bit/s	128Kbit	26876 KB	leaf	htb	0	[redacted]	**6.31**
2:18	64Kbit	115.6 Kbit/s	128Kbit	400610 KB	leaf	htb	0	[redacted]	**11.29**

Figura 2: Monitoreo de tráfico entrante con destino a los clientes.

En el gráfico de la *figura 1* se puede observar la topología de la red que se utilizó para las pruebas [3].

4.3. Monitores

Se implementó un monitor para los *qdiscs* del sistema en lenguaje php que permite obtener información estadística de funcionamiento en tiempo real. De esta manera, se pueden visualizar datos sobre cada uno de los *qdiscs* asociados a cada interfase del *router*. Para acceder a dicha información se puede poner este *script* disponible sobre un servidor web. El *script* se denomina: `qdisc_monitor.php`, y pasando el parámetro `dev=[dispositivo]` se puede obtener información sobre la interfase deseada. Ejemplo: `qdisc_monitor.php?dev=eth1` *figura 2*. Los datos obtenidos al ejecutar este programa son los siguientes (en el orden enunciado):

id de la clase, **rate** para el *qdisc* contenido en la clase, **actual rate** sobre el *qdisc*, **ceil** para el *qdisc*, **transfer** total de Kbytes transferidos, **type** tipo de nodo: [*root*, *node*, *leaf*], **qdisc** disciplina de encolamiento sobre la cola *qdisc*, cliente asociado al *qdisc*, porcentaje del ancho de banda total sobre el *qdisc*.

El proceso consta de la lectura de todos los *qdisc* asociados a la interfase que se desea consultar, volcando toda la información obtenida en una estructura jerárquica mediante arreglos. Luego se rea-

[3] Hay que considerar que el enlace con el cual cuenta la organización es del tipo PPPoE, lo cual implica que, si no se encuentra disponible, se desactiva la interfase virtual ppp0 levantada sobre el dispositivo de red público (interfase eth0). En dicho caso desaparecen los *qdisc* asociados a la interfase ppp0, teniendo que monitorear este suceso, para crear nuevamente los *qdiscs* cuando se reconecte la interfase ppp0.

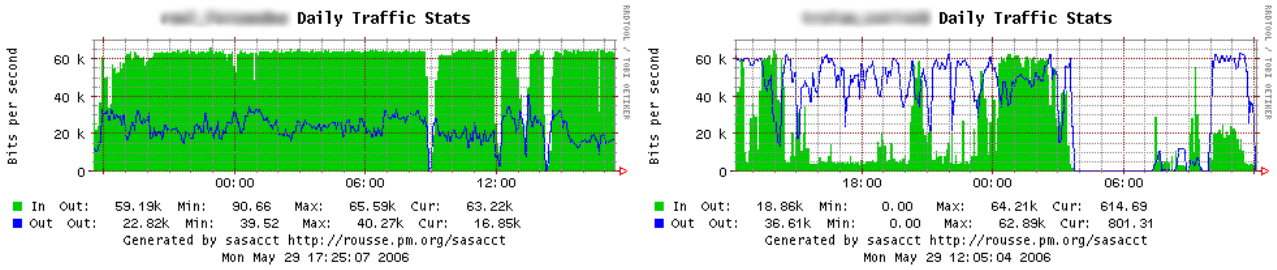

Figura 3: Cliente limitado a 64kbps.

liza un ordenamiento sobre la estructura según la clave en cada uno de los niveles del arreglo, para entonces consultar estos datos utilizando un único recorrido que muestra toda la información solicitada. Se marca en color verde el uso del ancho de banda permitido por debajo del *ceil* definido y por encima del *rate* para cada cliente. En rojo se indican aquellos casos en los cuales el *qdisc* está trabajando a una tasa de transferencia mayor al *rate* definido como máximo para el cliente, es decir su *ceil*. Estos casos son esporádicos y se deben a los intervalos de perturbación sobre la cola SFQ y a la forma en que se va regulando el tráfico de egreso de la interfase (utilizando el algoritmo HTB en la disciplina de encolamiento).

5. CONCLUSIONES

Los resultados de la experiencia obtenidos al implementar los conceptos enunciados fueron apropiados y el funcionamiento logrado fue satisfactorio. Se logró automatizar la generación de las reglas *tcng* y esto brinda la posibilidad al operador de cambiar sólo un archivo de configuración de acuerdo a las políticas de venta de la organización, sin tener que programar nuevos *scripts*.

La utilización de *tcng* evita la modificación de grandes *scripts* basados en tc cuando aumenta la cantidad de clientes en la red.

Se logró obtener información en una forma más comprensible a partir de los *qdiscs* de los dispositivos de red en el sistema y visualizarla en tiempo real.

Se aplicaron algunos conceptos básicos de QoS al priorizar cierto tipo de tráfico. En este sentido se pueden ampliar las mejoras según los requerimientos de la organización en el futuro, los conceptos mostrados pero ejemplifican las amplias posibilidades que brinda GNU/Linux como instrumento para la implementación de QoS y *traffic shaping* con distribuciones de software open source.

Se aplicó al *router* una protección global a través de iptables con políticas por defecto restrictivas y, a su vez, se protegió en forma básica a los equipos dentro de la intranet.

Se tomaron estadísticas para poder verificar los resultados obtenidos mediante gráficos apropiados (observar resultados en figura 3).

6. TRABAJOS FUTUROS

1. En el *firewall* se contemplarán las siguientes mejoras en futuras versiones:

   - Política por defecto DROP para la cadena OUTPUT y consideración de cada caso de servicios con permisos de salida.
   - Permitir el ingreso de paquetes a puertos de usuario, si previamente hubo una conexión saliente desde los mismos, ya que estamos implementando un SNAT y deseamos aceptar solamente conexiones entrantes, requeridas desde algún equipo de la intranet.

- Solamente se aceptarán equipos habilitados según las políticas de venta de la empresa, es decir, no cualquier cliente podrá tener acceso a servicios de Internet. Esto significa que las reglas de FORWARD del *firewall* especificarán explícitamente cada habilitación, para evitar que algún cliente se conecte a la red con un IP a elección. Se descartarán entonces paquetes de direcciones de IP internas no autorizadas.
- Se implementará un servidor DHCP para mejor la seguridad en cuanto a clientes autorizados a conectarse a la red.

2. Se detallarán formas de consultar el funcionamiento de los diferentes componentes utilizados para implementar todos los controles, explicación de comandos del tc e iptables.
3. Documentación sobre configuración y puesta en marcha procesos para generación de gráficos de monitoreo y estadística, usando las herramientas: mrtg, rrdtool y sasacct.
4. Mejoras del programa en C que se utiliza para invocar en modo consola al monitor de qdiscs.
5. Continuar con el desarrollo participativo de la herramienta destina a monitorear las clases y qdiscs de las interfases de red que cuenten con configuraciones basadas en *tc*. Para ello se creo un proyecto Open Source en el sitio `www.sourceforge.net` bajo la identificación: "Monitor for tc/tcng implementations".
6. Optar por priorizar el tráfico de servicios provenientes del equipo del *router* (POP3, SMTP, etc.) en una clase independiente, es decir para, establecer que el ancho de banda definido en el archivo `tcng_limits.txt` solamente se aplique para el tráfico proveniente desde Internet y no desde el servidor principal. Así los usuarios obtendrán una mejora considerable para los servicios locales, al mismo costo.

REFERENCIAS

[1] Werner Almesberger. Información sobre tcng. http://tcng.sourceforge.net.

[2] Werner Almesberger. Linux network traffic control - implementation overview, Feb 4 2001.

[3] Werner Almesberger, Jamal Hadi Salim, and Alexey Kuznetsov. Differentiated services on linux, Jun 1999.

[4] Oskar Andreasson. Iptables tutorial, v1.2.0, 2005.

[5] Leonardo Balliache. Differentiated service on linux howto, Aug 2003.

[6] Martin A. Brown. Traffic control howto, v1.0.1, Nov 17 2003.

[7] E. Crawley, R.Ñair, B. Rajagopalan, and H. Sandick. RFC 2386: A framework for QoS-based routing in the Internet, August 1998. Status: INFORMATIONAL.

[8] Federico Fapitalle, Javier Echaiz, and Jorge R. Ardenghi. Control de tráfico y administración de ancho de banda en linux con tcng.

[9] Bert Hubert, Gregory Maxwell, Remco van Mook, Martijn van Oosterhout, Paul B. Schroeder, and Jasper Spaans. Linux advanced routing and traffic control howto, v1.1.0, Jul 22 2002.

[10] LinuxNet. Información sobre iproute2. http://linux-net.osdl.org/index.php/Iproute2.

[11] LinuxNet. Información sobre iptables. http://www.netfilter.org/projects/iptables/index.html.

[12] Saravanan Radhakrishnan. Linux - advanced networking, Sep 30 1999.

[13] Chuck Semeria. Dsupporting differentiated service classes: Queue scheduling disciplines. Technical report, Juniper Networks, Dec 2001.